CIS 371 Web Application Programming

TypeScript VI

Lecturer: Dr. Yong Zhuang

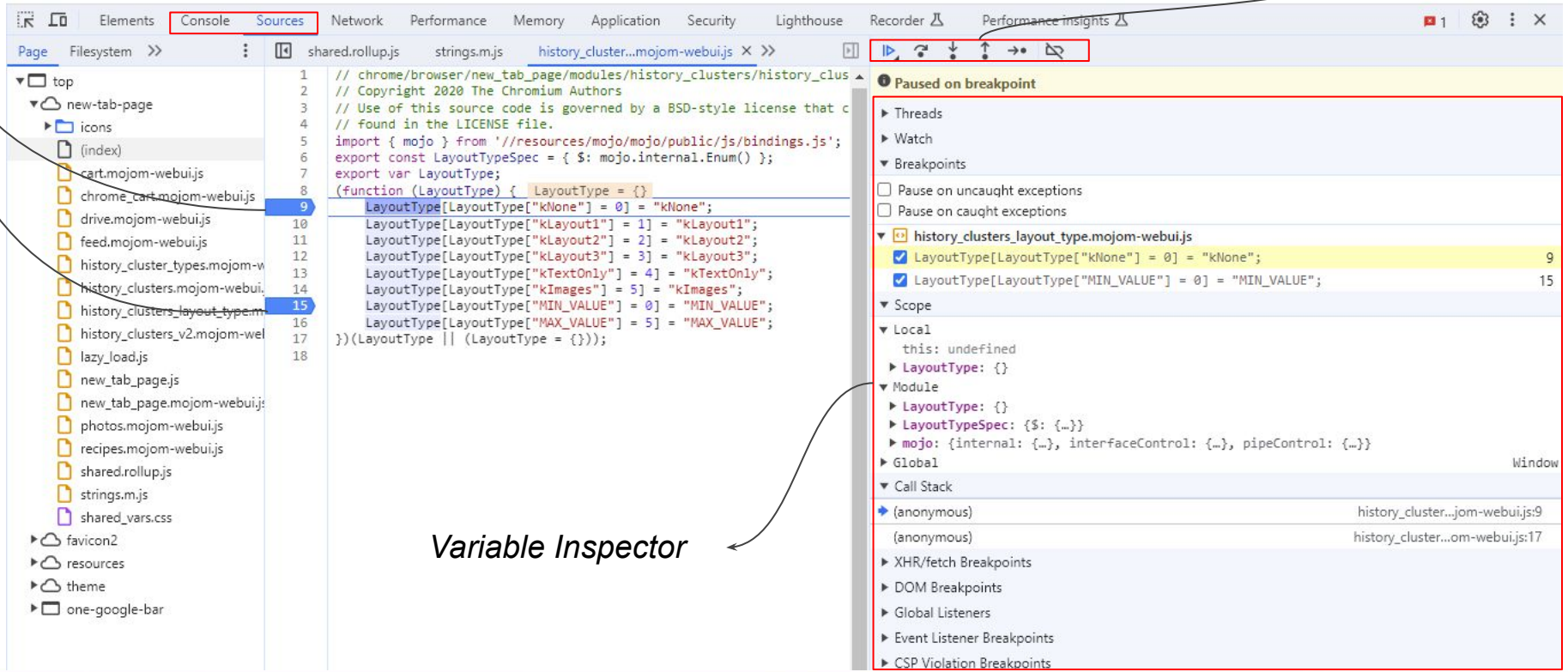
Based on the original version by Professor Hans Dulimarta.

Using TypeScript in Browser

Browser Debugger (Chrome)

Debugging breakpoints

Debugger Controls (step over, step into, ...)



Variable Inspector

Including JS code in HTML

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <script src="code1.js"></script>
    </head>
    <body>
        <!--    other HTML contents go here    -->
        <script src="code2.js"></script>
    </body>
</html>
```

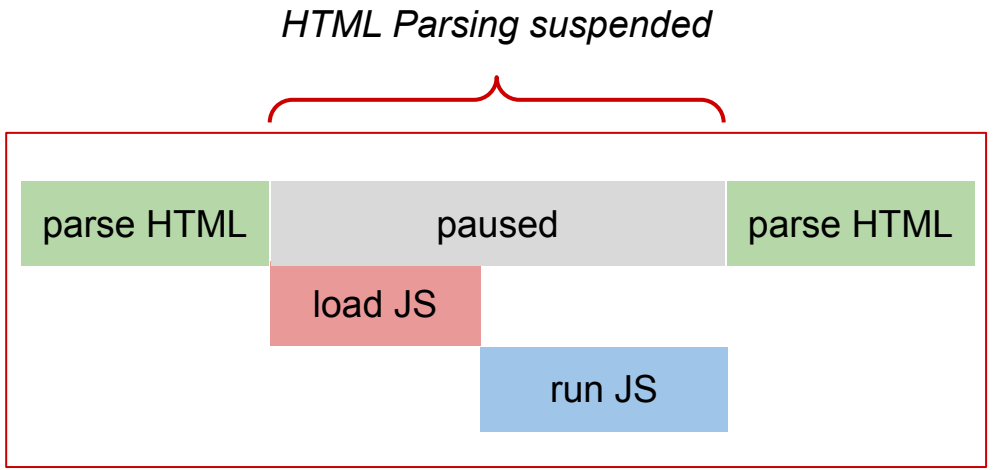
Scripts that do not modify page contents are placed in <head>

Scripts that do are placed towards the end of <body>

Script: Loading & Running

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>
    </head>
    <body>
        <!--    some HTML here    -->
        <script src="....js"></script>
        <!--    more HTML here    -->
    </body>
</html>
```

<script>



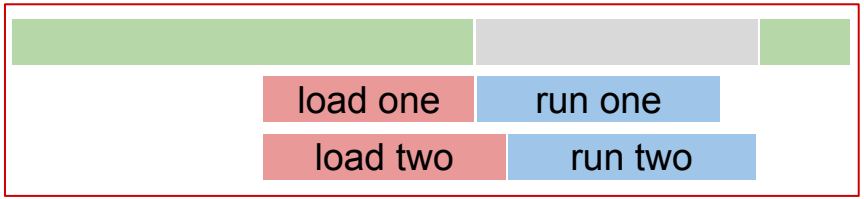
Defer vs. Async

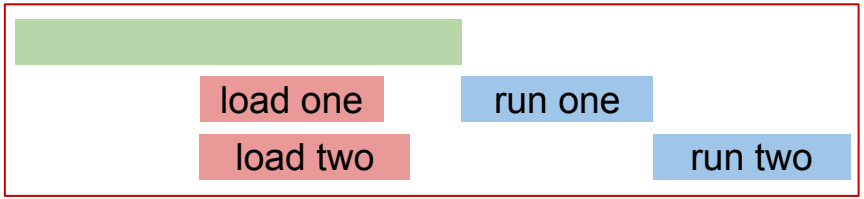
<script async>

```
<html>
    <body>
        <!-- some HTML here -->
        <script src="one" async></script>
        <script src="two" async></script>
        <!-- more HTML here -->
    </body>
</html>
```

<script defer>

```
<html>
    <body>
        <!-- some HTML here -->
        <script src="one" defer></script>
        <script src="two" defer></script>
        <!-- more HTML here -->
    </body>
</html>
```





When should I use what?

Typically you want to use **async** where possible, then **defer** then no attribute. Here are some general rules to follow:

- If the script is modular and does not rely on any scripts then use async.
- If the script relies upon or is relied upon by another script then use defer.
- If the script is small and is relied upon by an async script then use an inline script with no attributes placed above the async scripts.

async vs defer attributes

How to transpile TypeScript code so that it can run in a browser

TS <script> option #1: Babel

with babel-standalone

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
        <script type="text/babel" src="code1.ts"></script>
    </head>
    <body>
        <!--  HTML contents go here -->
        <script type="text/babel" src="code2.ts"></script>
    </body>
</html>
```

DO NOT use Babel standalone for production
Use transpiled JS for production with bundler (webpack, parcel, rollup, …)

TS <script> option #2: ParcelJS

```
npm init -y
npm install -save-dev parcel

# Create your-file.html with <script>
# Create one.ts and two.ts

npx parcel serve your-file.html

# Go to localhost:xxxx (in a browser)
```

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <script src="code1.js"></script>
    </head>
    <body>
        <!--  HTML contents go here    -->
        <script src="code2.js"></script>
    </body>
</html>
```

```
// one.ts
console.debug("Hello from one");
```

```
// two.ts
console.debug("Hello from two");
```

Browser predefined classes

- Classes associated with individual HTML tags

Tag	Class
<a>	HTMLAnchorElement
<body>	HTMLBodyElement
<button>	HTMLButtonElement
<img>	HTMLImageElement
<p>	HTMLParagraphElement
and many more …	

- Other classes: AudioBuffer, Bluetooth, ByteString, Promise, Request,...

Browser (Predefined) Objects

- Frequently used
  - screen: the computer screen occupied by the browser
  - document: the current HTML document that hosts the script
    - Provides functions for manipulating the DOM tree
  - window: the current window where the HTML doc is rendered
- Less frequently used
  - history: page visit history stack
  - localStorage: the browser persistent storage
  - location: the browser input box
  - and many more ...

```
for (const z in window) {
    if (typeof window[z] === "object") {
        console.log(z);
    }
}
```

Try this yourself

Browser window predefined functions

- alert(): show an info dialog on the browser
- addEventListener(): setup event listeners
- confirm(): show a yes/no dialog
- prompt(): show an input dialog
- setInterval(), setTimeout(): start a timer
- clearInterval(), clearTimeout(): reset existing timer
- ...
- and many more ...

```
for (const z in window) {
    if (typeof window[z] === "function") {
        console.log(z);
    }
}
```

Try this yourself

Complete documentations: **Web Windows API** (MDN: Mozilla Dev Network)

HTML Document CRUD methods/functions

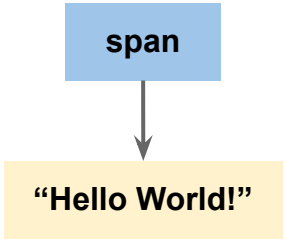
Create	`document.createElement()`, `document.createTextNode()`
Read	`___.getElementById()` // SINGULAR `___.getElementsByTagName()`, // PLURAL `___.getElementsByClassName()` // PLURAL `___.querySelector()` // SINGULAR: search by CSS selectors `___.querySelectorAll()` // PLURAL: search by CSS selectors
Update	`___.appendChild()`
Delete	`___.removeChild()`
and many more …	

```
for (const z in document) {
    if (typeof document[z] === "function") {
        console.log(z);
    }
}
```

Try this yourself

Create Text Nodes

```
<span>Hello world!</span>
```

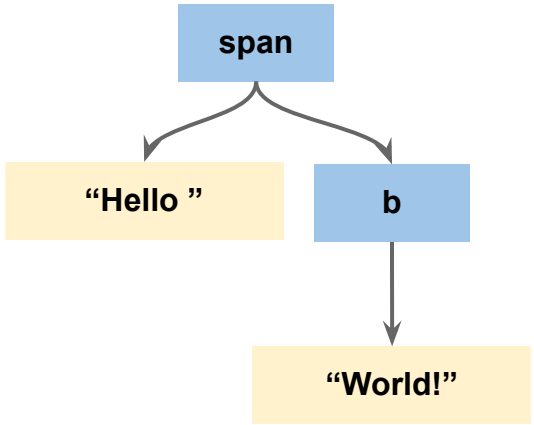



```
// Option 1
const spanParent = document.createElement("span");
const hello = document.createTextNode("Hello World");
spanParent.appendChild(hello);
```

```
// Option 2
const spanParent = document.createElement("span");
spanParent.innerText = "Hello World";
```

Add Multiple Children

```
<span>Hello <b>world!</b></span>
```




```
const spanTop = document.createElement("span");
const txt1 = document.createTextNode("Hello");
spanTop.appendChild(txt1);

const bChild = document.createElement("b");
bChild.innerText = "World";
spanTop.appendChild(bChild);
```

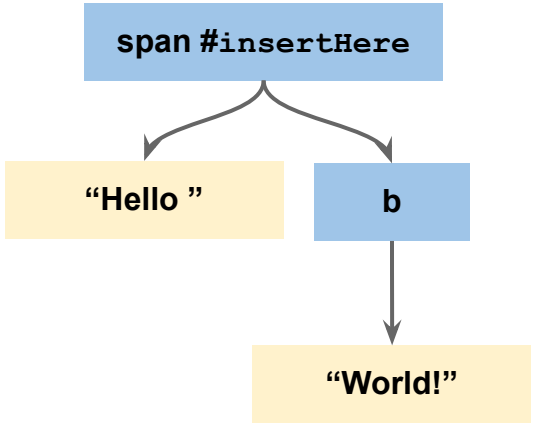
Insert Contents into Existing DOM

before

```
<span id="insertHere"> </span>
```

after

```
<span id="insertHere">
    Hello <b>world!<b>
</span>
```

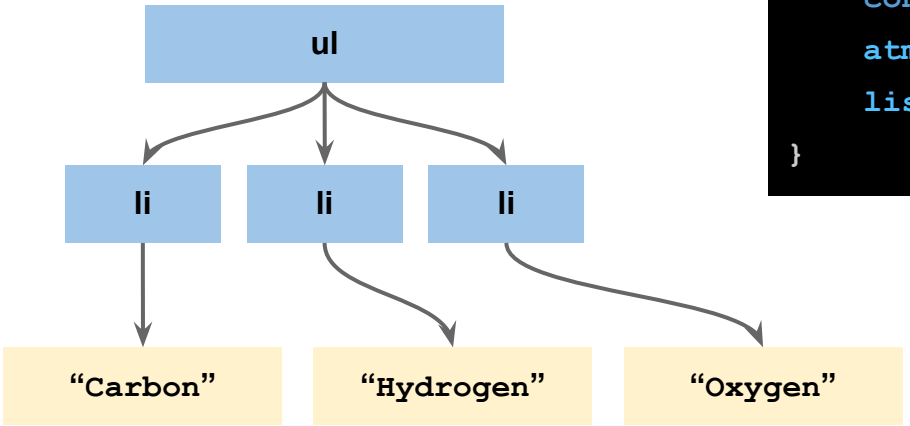



```
const spanTop = document.getElementById("insertHere");
const txt1 = document.createTextNode("Hello");
spanTop.appendChild(txt1);

const bChild = document.createElement("b");
bChild.innerText = "World";
spanTop.appendCHild(bChild);
```

Add Multiple Children from Array

```
<ul>
    <li>Carbon</li>
    <li>Hydrogen</li>
    <li>Oxygen</li>
</ul>
```




```
const atoms = ["Carbon", "Hydrogen", "Oxygen"]
const listTop = document.createElement("ul");

for (let a of atoms) {
    const atm = document.createElement("li");
    atm.innerText = a;
    listTop.appendChild(atm);
}
```

Setting attributes

```
<a id="intro" class="deepIndent noAds" href="http://go.org">
    Some text here
</a>
```

```
const sample = document.createElement("a");
sample.innerText = "Some text here";
sample.id = "intro"
sample.classList.add("deepIndent");
sample.classList.add("noAds");
sample.setAttribute("href", "http://go.org");
```

```
const sample = document.createElement("a");
sample.innerText = "Some text here";
sample.setAttribute("id", "intro");
sample.setAttribute("class", "deepIndent noAds");
sample.setAttribute("href", "http://go.org");
```

querySelector(): select ONE element

```
<body>
    <p class=”title”>Ice Cream Flavor:</p>
    <ul>
        <li>Death by Chocolate</li>
        <li>Mint Chocolate Chip</li>
        <li>Strawberry</li>
    </ul>
    <script src=”ice.ts”>
</body>
```

Ice Cream Flavors:

- Too much Chocolate
- Mint Chocolate Chip
- Strawberry

```
const item:Element = document.querySelector("ul > li");
#the first one will be returned
item.textContent = "Too much Chocolate";
```

querySelectorAll(): select MULTIPLE elements

```
<body>
    <p class="title">Ice Cream Flavor:</p>
    <ul>
        <li>Death by Chocolate</li>
        <li>Mint Chocolate Chip</li>
        <li>Strawberry</li>
    </ul>
    <script src="ice.ts">
</body>
```

```
let items:NodeListOf<Element>;
items = document.querySelectorAll("ul > li");
for (let flav of items) {
    if (flav.textContent.includes("Chocolate")){
        flav.textContent = flav.textContent + " (on sale)";
    }
}
```

Ice Cream Flavors:

- Death by Chocolate (on sale)
- Mint Chocolate Chip (on sale)
- Strawberry

CSS Selector and querySelector(All)

```
const q1 = document.querySelector("h2 + p");
q1.classList.add("red");   // Affect "First paragraph"
const q2 = document.querySelector("h2 ~ ol > li:first-child");
q1.classList.add("red");   // Affect "Strawberry"
const q3 = document.querySelector("li:last-child");
q1.classList.add("red");   // Affect "Barry Manilow"
```

```
<body>
  <h2>Some heading</h2>
  <p>First paragraph</p>
  <ol>
    <li class="fruit">Strawberry</li>
    <li class="device">Raspberry Pi</li>
    <li class="singer">Barry Manilow</li>
  </ol>
  <p>Second paragraph</p>
</body>
```

```
const pars = document.querySelectorAll("h2 ~ p");
for (let x of pars) {
    // Apply to "First paragraph" and "Second paragraph"
    x.setAttribute("___", "___");
}
const who = document.querySelectorAll("ol > li.singer");
for (let x of who) {
   // Apply to "Barry Manilow"
}
```

Using Timer

```
<body>
    <p>Ice Cream Flavor:</p>
    <ul>
        <li>Death by Chocolate</li>
        <li>Mint Chocolate Chip</li>
        <li>Strawberry</li>
        <li>Bluemoon</li>
    </ul>
    <script src="ice.ts">
</body>
```

Ice Cream Flavors:

- Too much Chocolate
- Mint Chocolate Chip
- Strawberry
- BlueMoon

2 seconds later

```
function choco() {
    const item:Element = document.querySelector("ul > li");
    item.textContent = "Too much Chocolate";
}
setTimeout(choco, 2000);
```

setTimeout(someFunc, delayInMillisec)

JavaScript Events

Source of Event	Events
Window	onload, onresize, onunload, ...
Document	onkeydown, onkeyup, onmousedown, onmouseup, onmouseenter, onmouseleave, ...
Input field	onblur, onfocus, onchange, ….
Button	onclick, ondblclick
Complete Reference: Event APIs	

Setting Up Event Handlers

- Which Event?
- Who is the event source?
  - Resize => window
  - Key presses => document
  - Load => document
  - Click => button, image, ….
  - Focus => input elements
  - Mouse => elements
- Details of the event object properties (MouseEvent, KeyboardEvent, ….).
  Refer to online API

```
function keyHandler(ev: KeyboardEvent): void {
    // put code here
}
function clickHandler(ev:MouseEvent): void {
    // put code here
}
document.addEventListener("keypress", keyHandler);
const myLogo = document.getElementById("myLogo");
myLogo.addEventListener("click", clickHandler);
```

addEventListener

inline event attributes

```
<button onclick="clickFunction()">Click Me</button>
```

```
function clickFunction() {
    // put code here
}
```

CodePen: Event Handling Demo

Counting Click